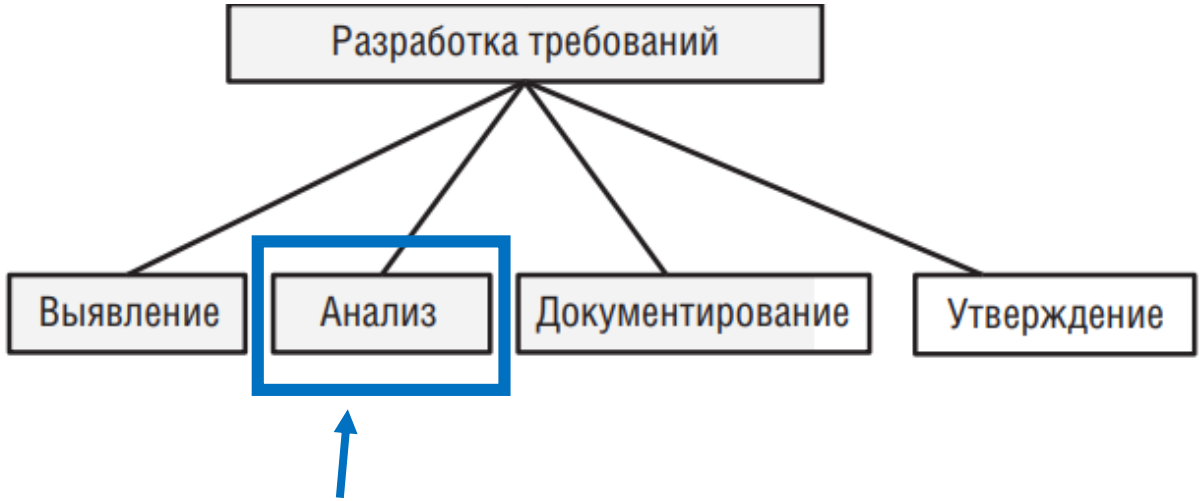
Анализ требований

Этот этап подразумевает получение более обширного и точного понимания всех требований и представление наборов требований в различном виде.



Основными действиями на этом этапе будут:

- анализ информации и отделение функциональных требований от нефункциональных, бизнес-правил, предполагаемых решений и другой информации;
- разложение высокоуровневых требований до нужного уровня детализации;
- выведение функциональных требований из информации других требований;
- распределение требований по компонентам ПО;
- согласование приоритетов реализации;
- понимание относительной важности атрибутов качества;
- выявление пробелов в требованиях или излишних требований, не соответствующих заданным рамкам.

Способы формирования пользовательских требований помимо Use Cases

Формат вариантов использования (Use Cases) лишен эмпатии к человеку, для которого создаётся программа. В сценариях использования его обычно называли презрительным и обезличенным «юзером».

Чтобы исправить эту ситуацию апологеты гуманности в парадигме **User Centered Design** сместили акцент на роль человеческого фактора. Появился метод персон, помогающий создать модели групп будущих пользователей и хоть как-то передать команде разработки понимание целей, задач и чаяний людей, для которых создаётся продукт.

Метод персон — инструмент, который позволяет исследовать целевую аудиторию, сегментировать ее и составить обобщенные портреты клиента — персоны. Помогает персонализировать маркетинговые предложения, выстроить коммуникацию, разработать интерфейс и пользовательский сценарий в продукте.

Это целая наука :о, если интересно смотри здесь
https://www.unisender.com/ru/glossary/chto-takoe-metod-person/

Метод персон

Алина

Возраст:

25

Пол:

Женский

Страна:

Россия

Семейное положение:

Незамужем

Пофессия:

Маркетолог

Работает на позиции маркетолога в малом бизнесе. Делает все, что так или иначе связано с маркетингом: рассылки, SMM, дизайн каталогов и корпоративных сувениров, отвечает за ивенты

Цели

Отправлять регулярные email-рассылки клиентам

Мотивация

Запускать рассылки легко и быстро, делать простые отчеты для шефа

Барьеры

Отсутствие готовых шаблонов и сложные настройки сервиса

Пользовательские истории (User Story)

- Пользовательская история - простое описание функциональности на языке пользователя, который говорит какую ценность или выгоду получит этот пользователь.
- Бизнесу это тоже выгодно потому что можно создать эту ценность и продать
- Пользовательские истории лучше Use Case-ов потому что показывают ценность.

Как **<роль или тип пользователя>,**
я хочу/могу **<выполнить действие или получить результат>,**
чтобы **<получить ценность>**

Как **"пациент стоматолога"**,
я хочу **"смотреть фильм в VR-очках во время сеанса лечения"**,
чтобы **"прием прошел приятно и время пролетело незаметно"**

User Story предложил **Кент Бек**: отец экстремального программирования, паттернов проектирования, JUnit и TDD
Интересная статья о нем на хабре по ссылке https://habr.com/ru/company/jugru/blog/580976/

Для чего применяется User Story?

- Для описания элементов *бэклога**
- Для лучшего понимания пользователей
- Для описания требований к продукту на понятном для всех языке
- Для вовлечения в процесс разработки пользователей и заинтересованных лиц
- Для построения **User Story Mapping**, которые помогают разбить проект на релизы

**Бэклог продукта — это перечень задач, расположенных в порядке важности, для команды разработчиков.*

Хорошие User Story соответствуют критериям INVEST

Стандартная формулировка: *Я, как <роль>, хочу <функция>, чтобы <ценность>* не содержит в себе всех критериев, важных для качественной разработки, поэтому в идеальном мире истории записывают, учитывая INVEST критерии.

INVEST - критерий правильности User story.

I.N.V.E.S.T. – это акроним, объединяющий шесть характеристик, которыми должен обладать Элемент Бэклога Продукта, чтобы соответствовать Критериям Готовности к Разработке.

Independent (независимая от других историй, то есть истории могут быть реализованы в любом порядке)

Negotiable (обсуждаемая, отражает суть, а не детали; не содержит конкретных шагов реализации)

Valuable (ценная для клиентов, бизнеса и стейкхолдеров)

Estimable (оцениваемая по сложности и трудозатратам)

Small (компактная, история должна быть краткой и ёмкой.)

Testable (тестируемая, имеет критерии приемки).

Критерий **Independent** (независимая от других историй)

Описание	"Симптомы"	"Лечение"
Пользовательская история может быть взята в работу и имплементирована независимо от других историй.	Прослеживается зависимость типа: •Пересечение •Зависимость порядка выполнения •Вложенность	Разбейте пользовательскую историю на более мелкие; предложите функциональность-заглушку или "hard-code", чтобы временно исключить зависимость.

"Плохой" пример	"Хороший" пример
Как клиент я хочу просмотреть список доступных достопримечательностей, настроенных Администратором, чтобы я мог решить, какие из них я хочу посетить.	Как клиент я хочу просмотреть список достопримечательностей, что бы решить какие из них я хочу посетить. Как администратор я хочу управлять списком достопримечательностей, что бы клиент смог просматривать актуальный список.

Критерий **Negotiable** (обсуждаемая)

Описание	"Симптомы"	"Лечение"
Пользовательские истории должны оставлять место для обсуждения.	Содержит слишком много деталей; предписывает конкретную технологию, подход к реализации или решение.	Пользовательские истории должны оставаться высокоуровневыми, с фокусом на конечной цели, а не на решении

"Плохой" пример	"Хороший" пример
Содержимое отчета необходимо экспортировать в электронную таблицу Microsoft Excel.	Содержимое отчета должно быть экспортировано в формате, позволяющем передать его в органы власти для дальнейшего аудита.

Критерий **Valuable** (ценная)

Описание	"Симптомы"	"Лечение"
Пользовательская история должна приносить пользу заинтересованным лицам (IRACIS).	История сосредоточена на технологии и преимуществах для разработчиков, а не на конечной цели.	Попытайтесь четко определить и сформулировать основную ценность (IRACIS: Increase Revenue, Avoid Costs, Improve Service, + Meet Regulations, Generate Information, etc.); объедините пользовательские истории;

"Плохой" пример	"Хороший" пример
Токен доступа должен инвалидироваться через 20 минут после начала сессии.	Система должна принудительно прервать сессию пользователя через 20 минут активности, в соответствии с политикой защиты данных клиента.

Критерий **Estimable** (оцениваемая)

Описание	"Симптомы"	"Лечение"
Историю можно оценить: сделать какое-то суждение о ее размере, стоимости или времени реализации.	Команда не может оценить историю :) Использование аббревиатур для конкретных доменов; ссылка на стандарты и процедуры предметной области; обобщения (например, *все*)	Предоставьте команде необходимые знания предметной области; укажите все недостающие детали; разделите на историю на спайк и основную историю.

"Плохой" пример	"Хороший" пример
Пользователь должен иметь возможность использовать все способы оплаты, доступные в Emerchantpay.	**Spike* для изучения API разных платежных систем.** **+ Пользовательские истории для каждого способа оплаты**

*Spike — это исследовательская задача, проводимая с целью помочь команде разработчиков лучше понять, что потребуется для реализации пользовательской истории.

Критерий **Small** (компактная)

Описание	"Симптомы"	"Лечение"
Пользовательская история не должна быть настолько большой, чтобы ее невозможно было планировать / приоритизировать с определенным уровнем точности (например, вписываться в спринт из 6-10 историй).	Оценка слишком велика	Разделяйте на более мелкие истории. Методы: SPIDR, CRUD, business rule variations, и т.п.

"Плохой" пример	"Хороший" пример
Как администратор я хочу получить доступ к статистике и сводным отчетам.	Разделить на более мелкие пользовательские истории для каждого конкретного типа отчета.

Критерий **Testable** (тестируемая)

Описание	"Симптомы"	"Лечение"
Проверяемая история - это история, для которой, при любых возможных входных данных, известно ожидаемое поведение системы.	Слова-триггеры: обобщения (все/ни один/любой/все комбинации/никогда), субъективные характеристики (хороший/плохой/лучший/легкий в использовании/интересный/быстрый/эффективный), некоторые наречия (быстро, безопасно, вовремя); неопределенные слова или аббревиатуры (и т.д., и/или, TBD); вероятностные понятия, аппроксимация.	Избегайте обобщений; переформулируйте субъективные понятия в измеримые величины.

"Плохой" пример	"Хороший" пример
Пользователь никогда не должен долго ждать, пока загрузится новая страница.	Новая страница загружается в течение двух секунд в 95% случаев.

Примеры хороших и плохих пользовательских историй

✗ Нет	✓ Да
Как пользователь, я хочу, чтобы приложение меньше зависало.	Как пользователь, который часто пользуется приложением по работе, я хочу, чтобы при зависании системы несохранённые файлы не терялись. **Что улучшили: прояснили боль пользователя.**
Как пользователь, я хочу общаться с друзьями, чтобы оставаться с ними на связи.	Как активная пользовательница соцсетей, я хочу пользоваться персонализированной лентой новостей, основанной на моих интересах и взаимодействии с друзьями, чтобы я не пропустила то, что для меня важно. **Что улучшили: описали функцию, которую можно разработать и протестировать.**
Как офисная сотрудница, я хочу полноценно питаться на работе, чтобы быть здоровой.	Как офисная сотрудница, я хочу иметь возможность заказывать полезную еду в офис, чтобы правильно питаться, когда у меня мало времени. **Что улучшили: проблеме не хватало конкретного решения — что именно поможет пользователю правильно питаться.**

https://practicum.yandex.ru/blog/chto-takoe-user-story-i-kak-napisat/

Связь требований с бизнес-целью и бизнес-требованиями

1. Зачем?

Бизнес-цель

- Привлечение клиентов
- Информирование клиентов
- Удержание клиентов
- Повышение лояльности и узнаваемости бренда или продукта
- Оптимизация процессов
- Повышение конкурентоспособности

Примеры для веб-приложений

2. Что?

Бизнес-требования. Цель достигается разными путями. И второй важный шаг при разработке требований как раз про выбор пути — **что конкретно мы будем делать**, чтобы прийти к цели.

3. Как?

Есть цель, есть путь ее реализации. Осталось разобраться с тем — **как мы это реализуем**: какие страницы будем показывать пользователям, в каком виде отобразим отчет для заказчика, как получим данные из другой системы, как будем хранить их у себя и так далее.

Все уровни требований связаны, следуют из требований верхнего уровня и являются основанием для требований более нижнего уровня

- Пользовательские требования
  - Функциональные требования
    - Нефункциональные требования
      - Ограничения

- Анкета должна содержать файл с фото, так как фото необходимо при оформлении документов — это **бизнес-требование.** А возможно, и бизнес-правило.
- Из бизнес-требования следует, что у пользователя должна быть возможность прикрепить фото к анкете — это **пользовательское требование**. То есть требование, описывающее действия пользователя.
- Получается, что система должна иметь функционал прикрепления фото к анкете — это уже **функциональное требование**, описывающее как должна работать система, чтобы выполнять исходное пользовательское требование.
- Будем хранить все фото в формате base64 в отдельной таблице в БД. Это — **нефункциональные требования**.
- Фото в очень хорошем качестве нам не нужно, а также мы не хотим покупать много памяти для сервера. Поэтому сделаем **ограничение** на размер загружаемого фото: не более 10Мб.

Карты пользовательских историй (User Story Mapping)

- ✓ помогают Agile-командам определять приоритетные элементы для разработки;
- ✓ визуально отображают, как отдельные составляющие продукта сочетаются друг с другом;
- ✓ направляют итеративный процесс разработки продукта;
- ✓ помогают учитывать интересы пользователей при обсуждении идей;
- ✓ расставлять характеристики продукта в приоритетном порядке.

Для чего применяется USM?

- для проектирования пользовательского опыта в продукте
- для определения границ MVP (минимальной работоспособной версии продукта) и планирования релизов на базе пользовательского сценария
- для формирования единого понимания пользователя у команды разработки и заинтересованных лиц

Как построить User Story Mapping?

вам потребуется:

- Инструмент визуализации: стикеры или электронный инструмент
- Владелец/пользователи продукта и команда разработки
- Представление о пользовательском сценарии (можно построенный CJM)
- 1-2 часа

Карта пользовательской истории отображает 3 типа действий разной степени детализации: **действия** (в общем смысле), **шаги и детали** (конкретные действия). Действия и шаги пользователя отображаются горизонтально в верхней части карты, а детали — вертикально, под соответствующими шагами в приоритетном порядке.

1. **Действия** представляют собой наиболее общие задачи, которые пользователь стремится выполнить в цифровом продукте, например, “Проверить баланс счета” или “Депонировать чек”. Количество таких действий будет различаться в зависимости от типа приложения или веб-сайта, который вы разрабатываете. При условии существования нескольких путей для различных типов пользователей они могут отражаться последовательно или параллельно. Поисковые исследования главных задач пользователей должны являться основой для создания этого уровня карты.
2. **Шаги** — это конкретные подзадачи, выполняемые пользователями для завершения действия, указанного выше. Они являются следующим уровнем после действий и также отображаются последовательно. Например, для действия “Депонировать чек” можно выделить следующие шаги: “Ввести данные мобильного депозита”, “Подписать чек”, “Сфотографировать чек”, “Внести депозит” и “Подтвердить депозит”.
3. **Детали** представляют собой третий уровень карты истории и описывают конкретные действия пользователей с наибольшей степенью детализации. Например, шаг “Войти в аккаунт” включает две отдельные детали: “Ввести имя пользователя” и “Ввести пароль”.

https://www.uprock.ru/articles/sozdanie-kart-polzovatelskih-istoriy-v-agile-proektah-3

Декомпозиция историй выполняется до тех пор, пока каждая из историй не будет отвечать критериям **INVEST** (независимость, обсуждаемость, ценность, возможность оценки сроков реализации, компактность, тестируемость).

1. Действия

Общие задачи пользователей в цифровом продукте

2. Шаги

Шаги, которые предпринимают пользователи для выполнения действия выше.

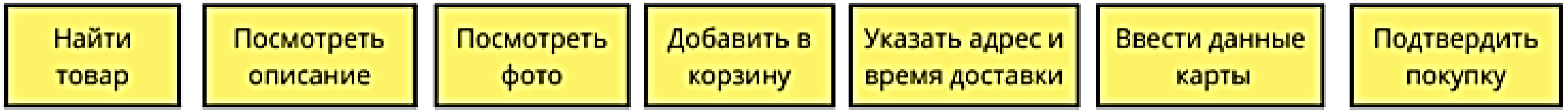
3. Детали

Конкретные взаимодействия, которые необходимы для выполнения шагов.

Рассмотрим USM на примере

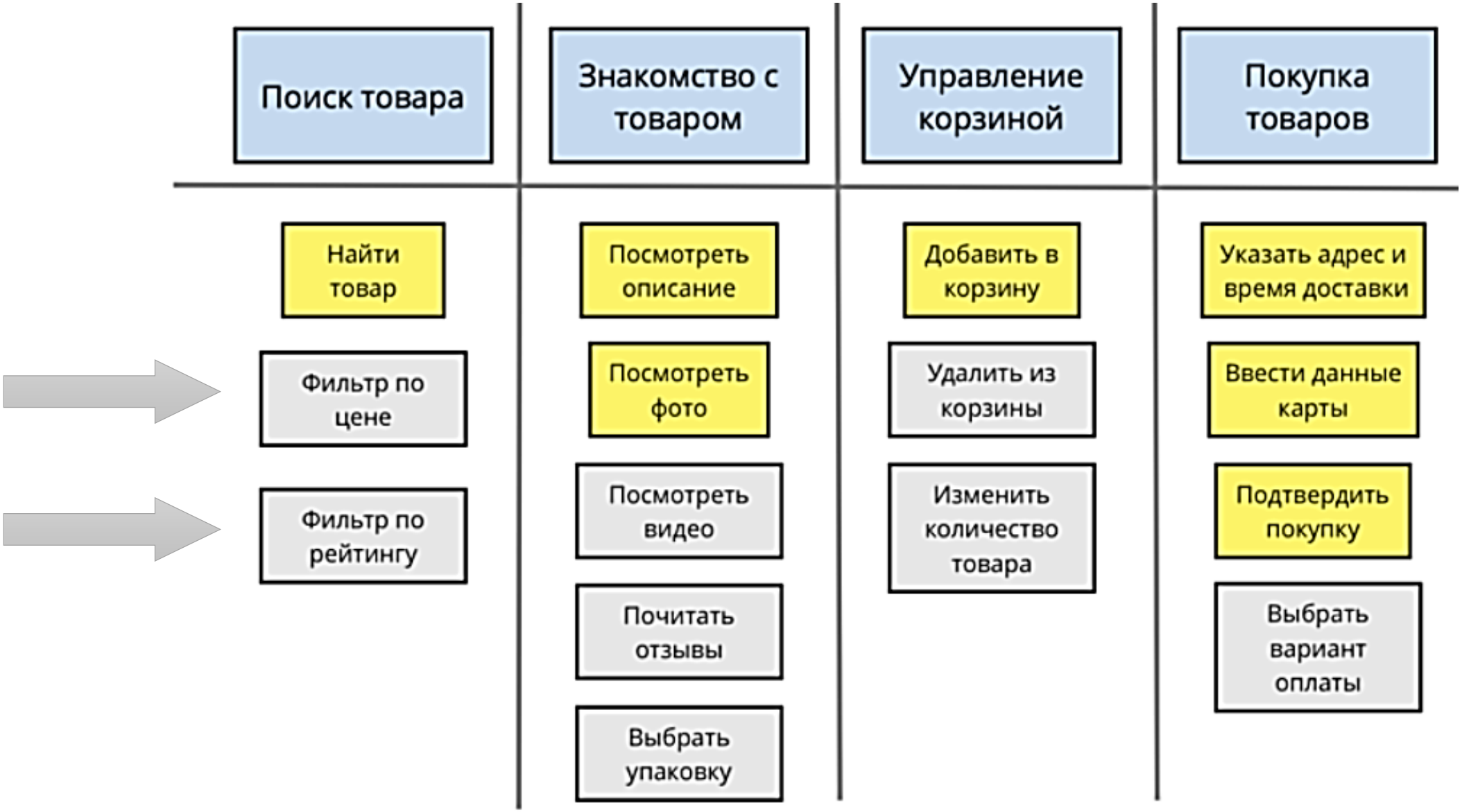
Магазин цветов решил запустить сайт. Визуализируем опыт клиентов с помощью техники USM.

1. Расскажите историю клиента по шагам

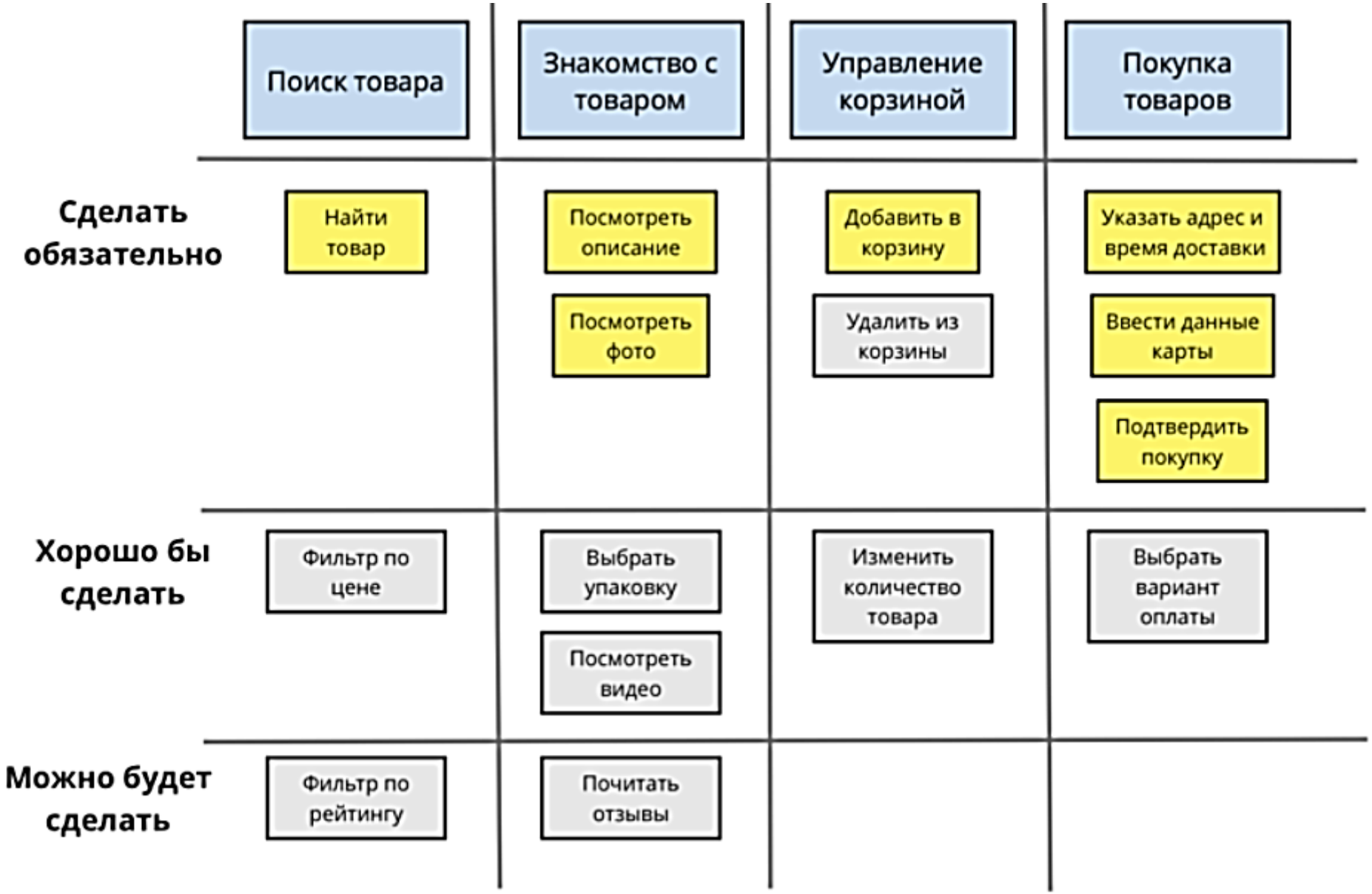


2. Сгруппируйте действия клиента в этапы

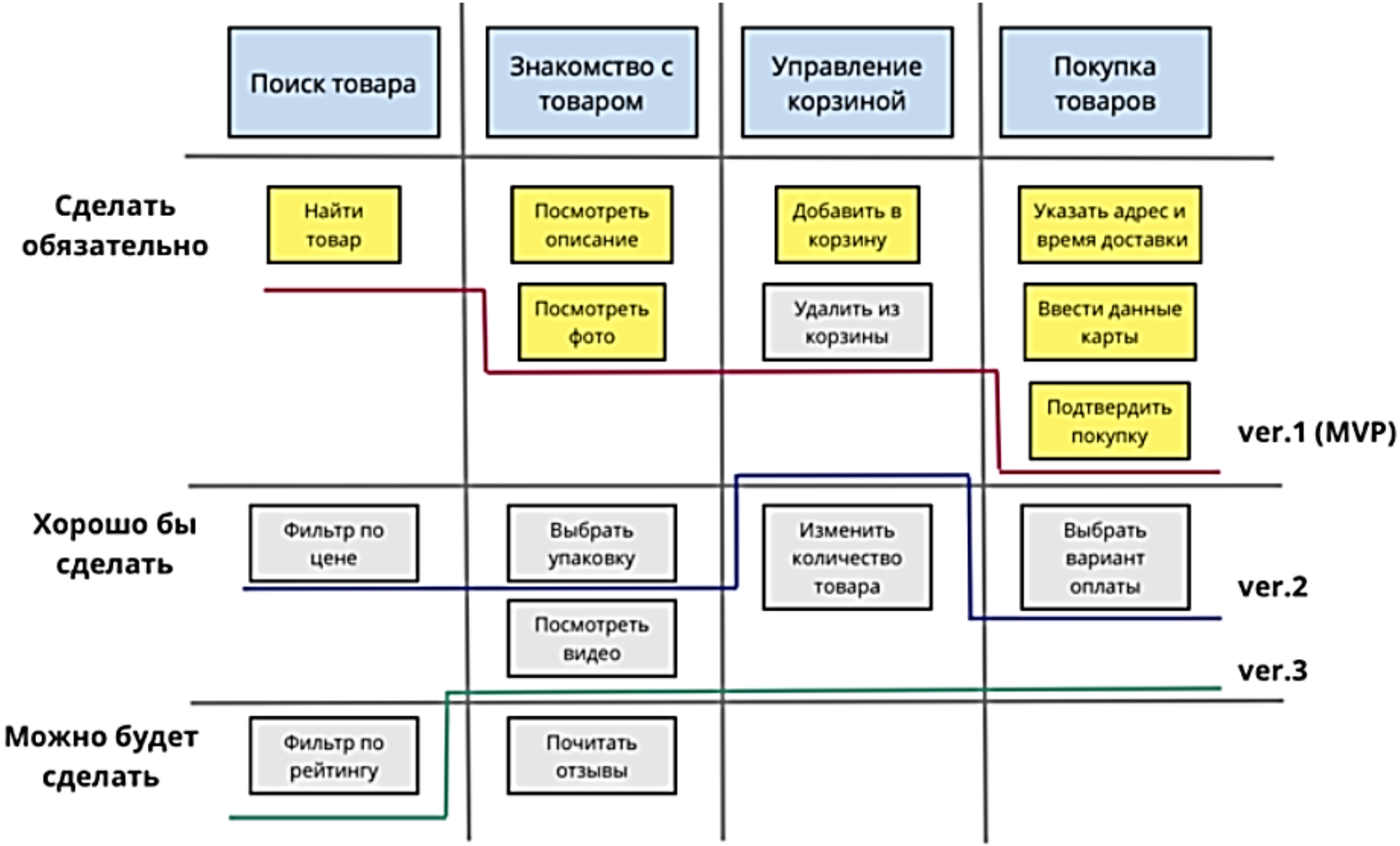
3. Заполнение пробелов в User Story



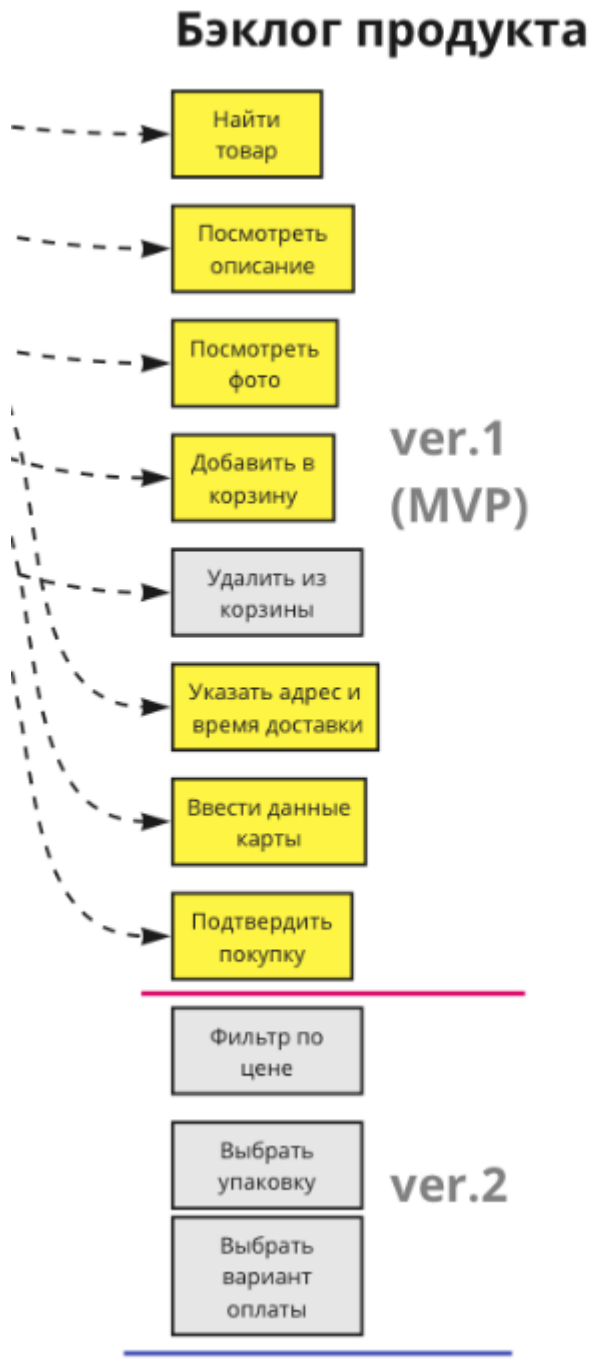
4. Приоритезируйте истории внутри каждого этапа пути



5. Выделите релизы



6. Итог- приоритизированный бэклог



User Story Mapping – мощный инструмент, позволяющий команде разработки за пару часов взглянуть на бэклог продукта глазами пользователя.

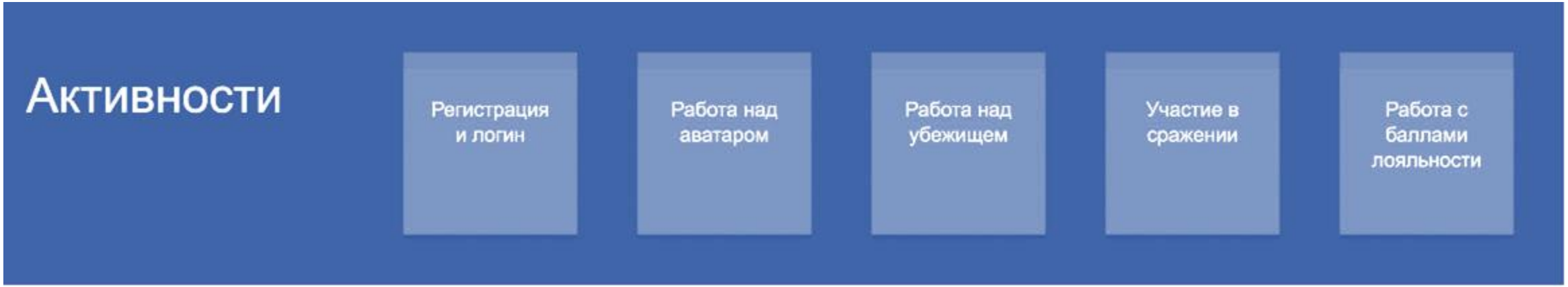
Во втором примере в качестве иллюстрации для составления USM будем использовать небольшую браузерную игру с программой лояльности какого-нибудь банка. В ней есть регистрация, игровой персонаж, его убежище, сражения и баллы лояльности.

https://sherer.pro/blog/user-story-mapping-i-funkcionalnaya-arxitektura/

Шаг 1. Выявляем активности

Сначала нужно собрать список основных **активностей** пользователей. В нашем примере игры их пять:

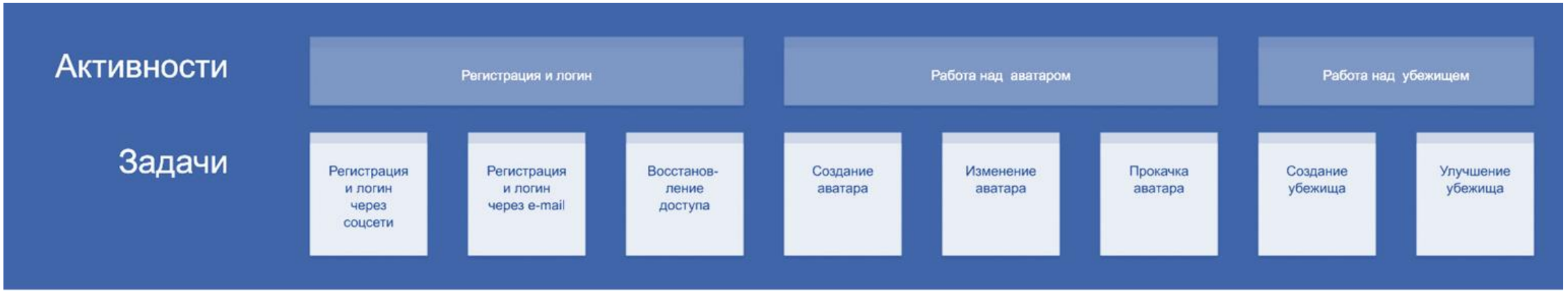
- Регистрация и логин;
- Работа над аватаром;
- Работа над убежищем;
- Участие в сражении;
- Работа с баллами лояльности.



Шаг 2. Разворачиваем активности в задачи

Раскладываем активности на **задачи**. Например, *активность* «Регистрация и логин» превращается в три *задачи*:

- Регистрация и логин через соцсети;
- Регистрация и логин через e-mail;
- Восстановление доступа.



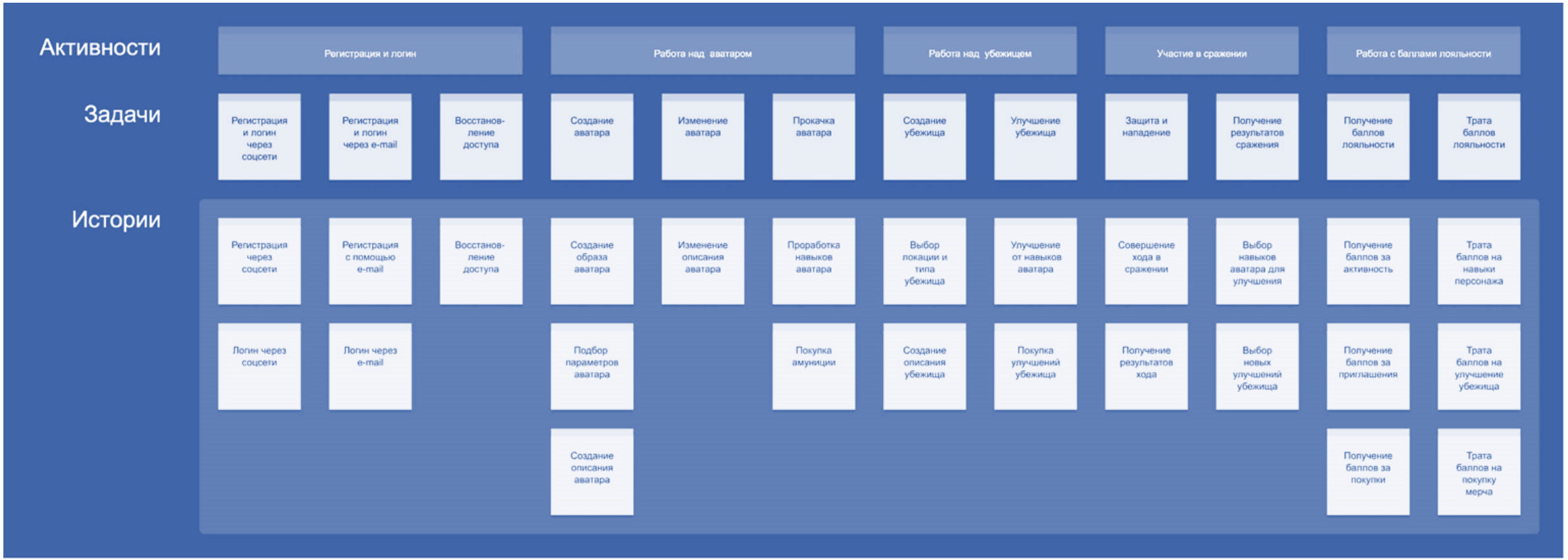
На уровне задач пока ещё нет отдельно «логина» или «регистрации». Это более глубокий уровень, задачи — это такие «категории» пользовательских историй.

Шаг 3. Задачи детализируем до конкретных историй

В нашем примера, ***задача*** **«Регистрация и логин через e-mail» разложилась на 2** ***истории***:

- Регистрация с помощью e-mail;
- Логин через e-mail;

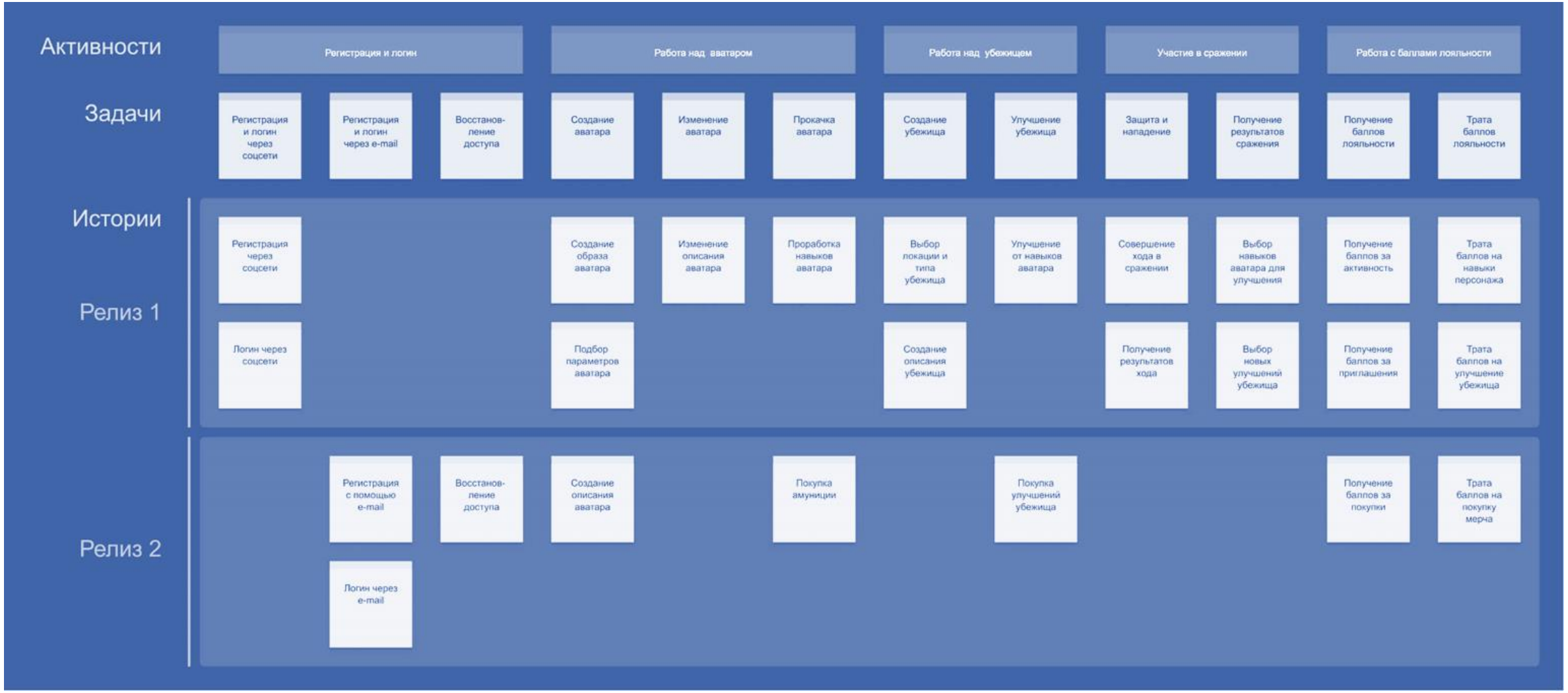
Истории (жёлтые карточки) — они расположены в порядке приоритета — сверху критичные, внизу наименее важные. **Истории — это функции будущей системы**. У них есть взаимосвязи и порядок.



https://sherer.pro/blog/user-story-mapping-i-funkcionalnaya-arxitektura/

Шаг 4. Делаем из приоритетов схему релизов

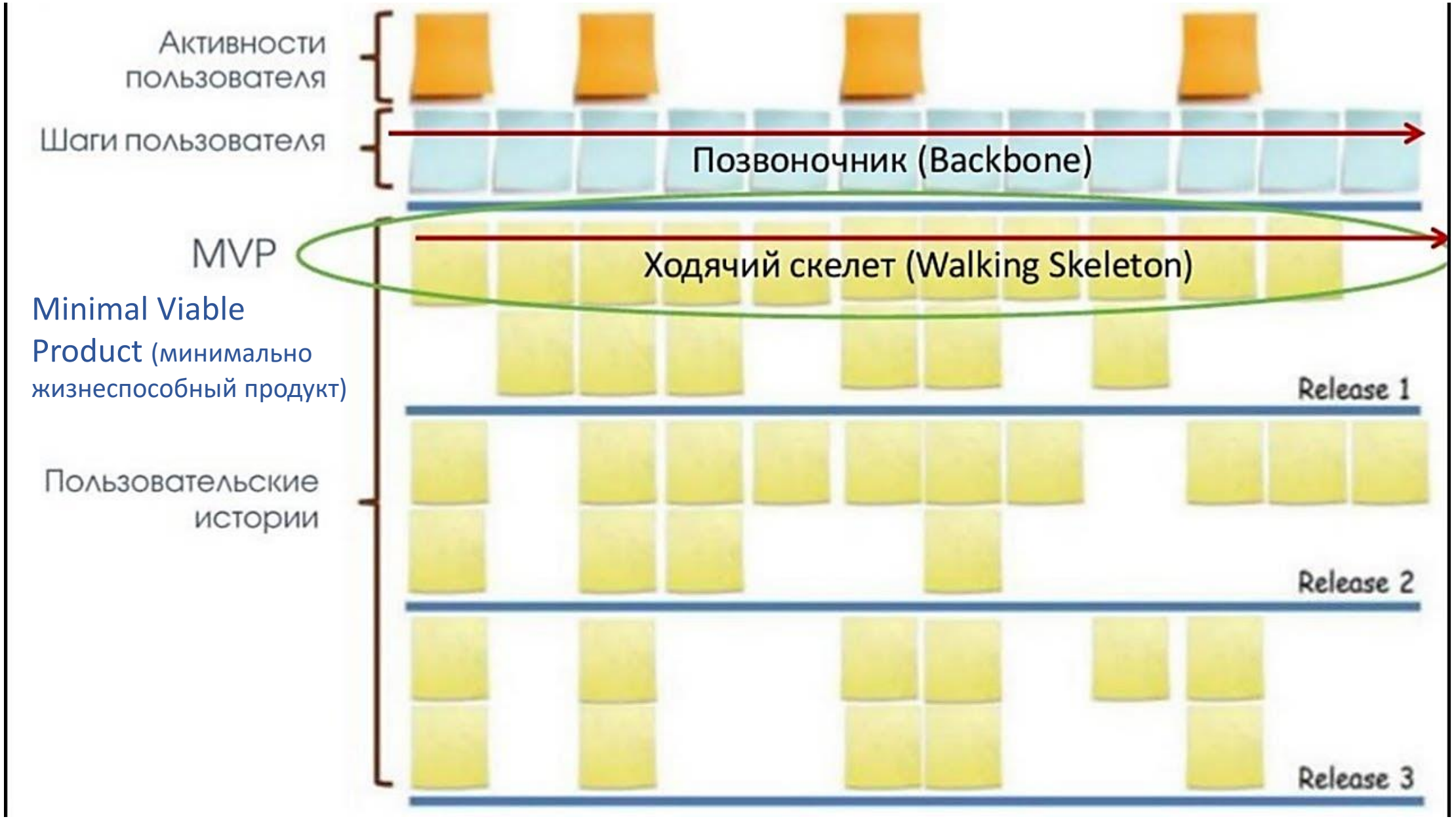
Переносим те истории, которые не критичны или требуют на этом этапе слишком много ресурсов, в следующий релиз. Например, нам проще сделать логин и регистрацию через соцсети — и регистрацию через e-mail мы отодвигаем на второй релиз.



Дополнительно: цветовая индикация и структура

Например, цветом стикеров можно помечать отдельные категории событий/действий, дополнительную важность (например, для бизнеса), типы пользователей и так далее.

Верхняя часть, с активностями и задачами, называется «backbone» (позвоночник). Нижняя часть, истории, называется «тело».



Minimal Viable Product (минимально жизнеспособный продукт)

Плюсы User Story Mapping :

Улучшение коммуникаций внутри команды. У каждого из нас уникальное видение чего-либо. В обычной жизни это хорошо, но для людей, работающих над одним продуктом, картина должна быть общей. User Story Mapping — простой способ показать команде, над чем она работает и к каким результатам должна прийти совместными усилиями.

Фокус на пользователях. В глазах потребителей ценный продукт = закрытые потребности. Если это игнорировать и работать только над функционалом, то ценность продукта будет сведена к нулю. Техника User Story Mapping фокусирует внимание команды на потребностях конечных пользователей, а не только на функционале.

Максимальная визуализация. Карты пользовательских историй раскрывают User Flow — визуальное представление того, какие действия потребитель совершает, чтобы достигнуть цели. Это позволяет разработчикам не упустить важных шагов, а пользователям — получить продукт, который закроет их потребности.

Минусы у User Story Mapping тоже есть. Как минимум — сложность организации. Если продукт объемный, составить бэклог за стандартные 2–3 часа невозможно. Проект придется разбить на несколько отдельных задач и выстраивать USM по каждой в отдельности.

Руководство к своду знаний по бизнес-анализу BABOK приводит следующее определение:

Приоритизация требований – это ранжирование требований, чтобы определить их относительную важность для стейкхолдеров через выставление приоритета – оценки, отражающей ценность требования или очередности реализации.

Приоритезация требований считается основополагающей деятельностью для любого типа разработок: водопада или Agile.

Целью приоритезации требований является:

- ✓ Разработка наиболее важных требований в первую очередь;
- ✓ Распределение усилий, направленных на разработку, с целью оптимизации прибыли на инвестиционный капитал.

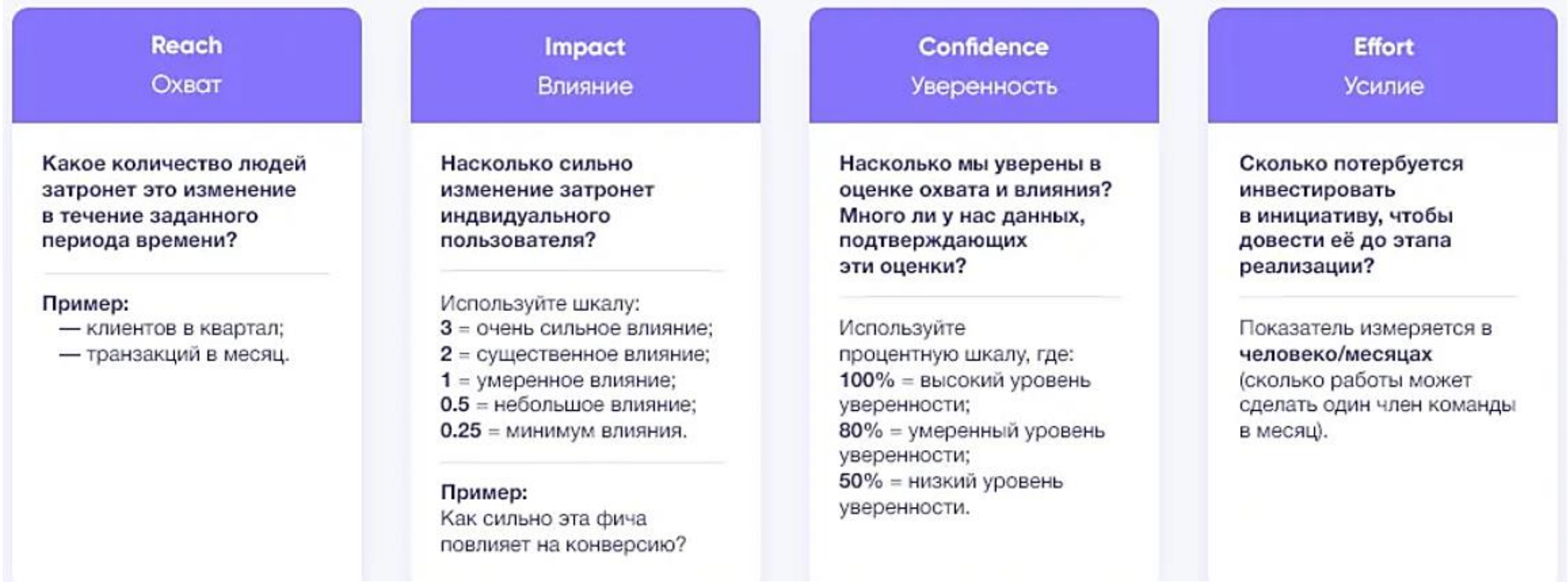
Приоритет может выражаться количественно, например, по шкале 1-10 или качественно: высокий, средний, низкий.

Наиболее распространенные методы приоритизации требований:

- ✓ **RICE** (Reach — охват, Impact — влияние, Confidence — достоверность, Effort — усилия)
- ✓ **ICE** (Impact — влияние, Confidence — уверенность, Ease — легкость реализации)
- ✓ **Value vs Effort** («ценность против усилия»)
  Модель Кано
- ✓ **Story mapping** («карта историй»)
- ✓ **MoSCoW**
- ✓ **Opportunity scoring** («оценка возможностей»)
- ✓ **Product tree** («дерево продукта»)
- ✓ **Cost of delay** («стоимость задержки»)
- ✓ **Buy a feature** («купи функцию»)
- ✓ **Weighted scoring** («взвешенная оценка»)
- ✓ **WSJF** (Weighted Shortest Job First, «сначала — более важная и короткая работа»)

Подробнее о методах по ссылке:
https://vc.ru/marketing/274778-12-metodov-prioritizacii-produktovyh-celey-rice-wsjf-kano-i-prochie

RICE Эта система оценки измеряет каждую фичу или инициативу по четырем параметрам: охват (reach), влияние (impact), достоверность (confidence) и усилия (effort).



$$\frac{\text{Охват} \times \text{Влияние} \times \text{Уверенность}}{\text{Усилие}} = \text{Оценка по RICE}$$

Техника приоритизации Agile Backlog: MoSCoW

MoSCoW — это метод расстановки приоритетов, обычно используемый в Scrum.

Аббревиатура MoSCoW определяет четыре категории приоритетов:

must-have — обязательно должно быть сделано;

should-have — должно быть сделано;

could-have — могло бы быть сделано;

won't-have — не в этот раз

M	Must	Должны
O	O	
S	Should	Хотелось бы
C	Could	Могли бы быть
O	O	
W	Won’t	Не в этот раз

Must – то, что необходимо сделать в любом случае. Без выполнения этих задач продукт не будет работать в принципе.

Should – не самые важные требования, но они тоже должны быть выполнены. Естественно, после реализации «must».

Could – желательные требования, которые можно сделать, если останется время и будут ресурсы.

Won't – требования, которые можно проигнорировать или перенести на следующие релизы без вреда для продукта.